Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Position control of a mobile robot using reinforcement learning

G. Farias * **G. Garcia** ** **G. Montenegro** * **E. Fabregas** ***
S. Dormido-Canto *** **S. Dormido** ***

* *Pontificia Universidad Católica de Valparaíso, Av. Brasil, 2147, Valparaíso, Chile (e-mails: gonzalo.farias@pucv.cl, guelis.montenegro@gmail.com).*
** *Ocean and Mechanical Engineering, Florida Atlantic University, Boca Ratón, FL 33431, USA. (e-mail: garciag@fau.edu).*
*** *Departamento de Informática y Automática, Universidad Nacional de Educación a Distancia (UNED), Juan del Rosal, 16, 28040, Madrid, Spain (emails: {efabregas,sebas,sdormido}@dia.uned.es).*

Abstract: Robotics has been introduced in education at all levels during the last years. In particular, the application of mobile robots for teaching automatic control is becoming more popular in engineering because of the attractive experiments that can be performed. This paper presents the design, development, and implementation of an algorithm to control the position of a wheeled mobile robot using Reinforcement Learning in an advanced 3D simulation environment. In this approach, the learning process occurs when the agent makes some actions in the environment to get some rewards. Trying to make a balance between the new information of the environment and the current knowledge about it. In this way, the algorithm is divided into two phases: 1) the learning stage, and 2) the operational stage. In the first stage, the robot learns how to reach a known destination point from its current position. To do it, it uses the information of the environment and the rewards, to build a learning matrix that is used later during the operational stage. The main advantage of this algorithm concerning traditional control algorithms is that the learning process is carried out automatically with a recursive procedure and the result is a controller that can make the specific task, without the need for a dynamic model. Its main drawback is that the learning stage can take a long time to finish and it depends on the hardware resources of the computer used during the learning process.

Keywords: Control Education, Mobile Robot, Position Control, Reinforcement Learning.

1. INTRODUCTION

Robotics is a suitable platform to teach control engineering fundamentals because it provides a lot of interrelated elements, for example, mechanics, electronics, control, programming and so on. As autonomous machines, robots are a whole example of a closed control loop: 1) sensors, 2) control unit and 3) actuators. That is why many universities use robots to teach basic concepts in control (Fabregas et al. (2017); Farias et al. (2019)). In this context, mobile robots represent a good tool to carry out laboratory practices with students in virtual environments. This work proposes the use of the Khepera IV robot library into V-REP simulator (Farias et al. (2017)). The simulator is a versatile and scalable framework for creating 3D simulations in a relatively short period of time. It allows designing and performing experiments before using real Khepera IV robots.

One of the most one basic practice that can be performed with mobile robots is called position control. This experiment is known as point stabilization or position control of a nonholonomic mobile robot. It deals with the problem of carrying a robot from its current position to a known destination point (Siegwart et al. (2011); Tzafestas (2018)). This experiment has been studied and used for teaching during the last years, using different traditional linear and nonlinear control techniques, such as: nonlinear model predictive control (Rezaee (2017)), back-stepping control with asymptotic stability (Dumitrascu et al. (2011)), continuous time-varying adaptive controllers (González et al. (2010)), PID controllers (Villela et al. (2004); Fabregas et al. (2020)) and much more. Other approaches to deal with this problem have implemented machine learning techniques (ML), for example, genetic algorithms (Caceres et al. (2017)), neural networks (Mohareri et al. (2012)), and fuzzy logic applied to ML (Omrane and Masmoudi (2016)), etc.

These two paradigms of control have in common that they can solve this kind of problem but using two different approaches. The traditional control techniques, in general, design a controller by obtaining its parameters. These parameters can be adjusted using different existing techniques. The result is an algorithm that takes the inputs and calculates the action to make the control most efficiently. On the other hand, the methods of artificial intelligence, in general, obtain a control law from data or an agent learns how to solve a specific problem adjusting automat-

ically the internal parameters. Most of these algorithms need a learning stage, where the agent learns to solve the problem, but this process can take a long time, depending on the complexity of the task to solve. From this group, Reinforcement Learning sticks out as it is not dependent on any prior dynamic model of the system.

The main goal of this paper is the design, development, and implementation of an algorithm to control the position of a wheeled mobile robot using Reinforcement Learning (RL), based on Busoniu et al. (2017); Sutton and Barto (2018). This is an area of machine learning where an agent interacts with the environment to get some rewards for its actions. From this interaction, the agent must learn how to make a specific task finding a balance between the new information obtained from the environment and its current knowledge of it. In this case, the robot must learn how to reach a known target point from its current position.

The process is divided into two phases: 1) Learning stage, and 2) Operational stage. During the first stage, the agent is trained in simulation to obtain the learning matrix, which is the result of this phase. During the second stage, the robot uses this matrix to carry out control of its position. The main advantage of this approach with respect to the traditional control algorithms is that the learning process is carried out automatically in a recursive procedure in a computer and the result is a controller that can make a specific task. No dynamic model of the plant is ever needed. This process can be carried out in simulation initially, and then the controller is directly implemented in the real robot for operation, or for a reduced time fine tuning learning stage to capture features not included in the emulated model. In this way, the damages to the robot can be avoided during the learning stage. Its main drawback is that the learning stage can take a long time to finish and it depends on the hardware resources of the computer used to train and obtain the model. This experiment can be easily added to the pedagogical platform for performing hands-on experiments with advanced mobile robots (Farias et al. (2019)).

The remainder of the paper is organized as follows: Section 2, presents some theoretical aspects about the Reinforcement Learning approach; Section 3 describes its application to mobile robot position control; Section 4 shows and discusses some simulation results of this research, and Section 5 presents the main conclusions and future works.

2. REINFORCEMENT LEARNING

RL is an area of machine learning concerned with an agent that interacts with the environment to receive cumulative rewards for its actions.

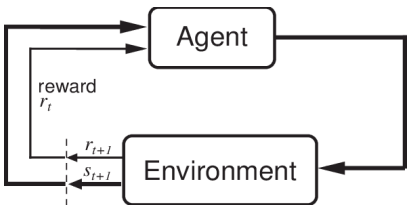

Fig. 1. A typical framing of a RL scenario.

Figure 1 shows a typical RL scenario, where an agent makes actions in the environment, which are interpreted into a reward and a representation of the next state, which is feedback into the agent Jaksch et al. (2010).

RL is one of the three paradigms of machine learning: 1) Supervised learning, 2) Unsupervised learning and 3) Reinforcement learning. This algorithm is focused on finding a balance between the exploration of unknown territory i.e. new information of the environment and the utilization of its current knowledge about it (Busoniu et al. (2017); Sutton and Barto (2018)).

2.1 Recursive algorithm Q-learning

The environment is typically formulated as a Markov Decision Process (MDP), where many reinforcement learning algorithms within this context are based on dynamic programming techniques, see White (2001); Busoniu et al. (2017). The deterministic MDP defined by $x_{k+1} = f(x_k, u_k)$ and the reward function $r_{k+1} = \rho(x_k, u_k)$ constitute the dynamics of the algorithm, with $u_k = l(x_k)$ a static control law to be determined in a recursive procedure, and where x_k is the state of the system. In this application, the reward function is known and given by the designer, and the MDP function is shaped by the plant itself, no model is needed during the fine-tuning of the controller. For this purpose, an infinite-horizon return with discount $0 < \gamma < 1$ is defined by:

$$\sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, l(x_k)), \tag{1}$$

representing the accumulated discounted rewards into the future from the current state x_0 by applying the policy l. The discount factor penalizes future rewards. From this definition, the action-value function:

$$\begin{aligned} Q(x_0, u_0) &= \rho(x_0, u_0) + \sum_{k=1}^{\infty} \gamma^k \rho(x_k, l(x_k)) \\ &= \rho(x_0, u_0) + \gamma \sum_{k=0}^{\infty} \gamma^k \rho(x_{k+1}, l(x_{k+1})), \end{aligned} \tag{2}$$

is defined, that allows the assessment of the goods for an agent to perform the action (u_0) while in the state x_0 and then continue using policy l. The optimal policy is then obtained by maximizing the future reward through $l^*(x) = \gamma \max_v Q(x, v)$. This control law achieves the optimal action-value function $Q^*(x, u)$ which can be rewritten recursively, constituting a Bellman equation:

$$\begin{aligned} Q^*(x_k, u_k) &= \rho(x_k, u_k) + \gamma \max_v Q^*(x_{k+1}, u_k) \\ &= \rho(x_k, u_k) + \gamma \max_v Q^*(f(x_k, u_k), v), \end{aligned} \tag{3}$$

A Bellman equation characterizes the Principle of Optimality by stating that future optimal control values are defined only by the state values from where they are computed, independently of past control actions and states. In a discrete framework, this equation allows a backward-in-time recursive computation to solve for optimal policy, like in dynamic programming. The Bellman equation characterizes Q^* and declares that the optimal value of action u took while in state x equals then to the sum of

the immediate reward and the discounted optimal value obtained by the best action in the next reached state by previously using u. Watkins (1989) and Watkins and Dayan (1992) developed a forward-in-time model-free iterative algorithm, called Q-learning, that asymptotically converges to Q^* as time k goes to infinity. This procedure incrementally populates the action-value function during operation of the system without the need for any back-in-time calculation, and the need for any modeling, with the system materializing the function f. After the transition from x_k to x_{k+1} using u_k, the following action-value function can be updated:

$$Q^{i+1}(x_k, u_k) = Q^i(x_k, u_k) + \alpha(\rho(x_{k+1}, u_k) + \gamma \max_v Q^i(x_{k+1}, v) - Q^i(x_k, u_k)) \quad (4)$$

with $(0 < \alpha < 1)$ a learning rate. This Q-learning relation can be seen as coming from a discrete low pass filter defined by the accumulated knowledge $Q^i(x_k, u_k)$ and the incoming new information, the temporal difference $TD^i(x_k, u_k)$ as:

$$Q^{i+1}(x_k, u_k) = \alpha TD^i(x + k, u_k) + (1 - \alpha)Q^i(x_k, u_k) \quad (5)$$

with $TD^i(x_k, u_k) = \rho(x_{k+1}, u_k) + \gamma \max_v Q^i(x_{k+1}, v) - Q^i(x_k, u_k)$. Initial condition of Q can be set to zero, i.e. $Q^0 = 0$ or to the reward function.

2.2 Application of Q-learning

In this paper, the application of the algorithm is divided into two stages: 1) a learning process, and 2) an operational process. Both phases are performed with an emulated version of the robot f, described in the next section.

For a feasible application of equation (4), both states and control variables are finitely discretized covering the working state space, replacing the function Q by a matrix Q with a fixed amount of entries.

Learning stage. This stage is characterized by a limited degree of randomness involved in the selection of the current action to be taken. As the robot is learning as it moves in a flat surface, the learning process is set to constantly be shifting between a purely random control action, and a deterministic one, based on previously acquired knowledge of the environment, to avoid the robot to get away from the initial position, and/or the target position. The deterministic version is defined as $u_k = \max_v Q^i(x_k, v)$, while the random control is done by simply randomly selecting a control action among the available ones. Note that a system is a white-box approach because we know the model of the robot. If we were considered a black-box approach (unknown model) the learning stage had to be done with the actual robot, which could imply a must larger training time and might damage the actuators (motors).

Operational stage. After the learning process, where the matrix Q has been populated, the operational phase can be entered. The analysis of the best criteria for stopping the learning process is not covered in this work, an empirical one was used, although this topic is of high importance, as it is directly related to the goodness of the convergence of the action-value matrix to its optimal one. The control in this stage is determined by the same deterministic control used during the learning phase, with the difference that the action-value matrix is not changing and set fixed as it ended up in the learning stage.

3. MOBILE ROBOT POSITION CONTROL WITH RL

3.1 Position control experiment

This experiment consists of driving a mobile robot from point C (current position of the robot) to point Tp (target point), by manipulating its angular (ω) and linear (V) velocities. Note that these velocities are then transformed into speeds for the left and right motors as the robot is a double wheeled one. Figure 2 shows the variables involved in this experiment.

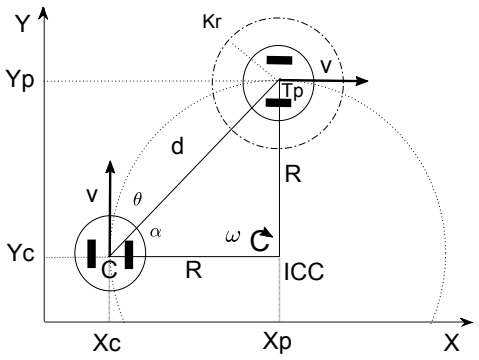

Fig. 2. Position control experiment of a differential robot.

This problem has been widely studied during the last years as the kinematic behavior of these robots may seem deceptively simple, but nonholonomic constraints introduce a challenging problem in the designing of the control law. This has been explained in more detail in some of the author's previous works, Fabregas et al. (2016) and Fabregas et al. (2020). In regular motion, the differential robot describes a circular trajectory of radius R with center ICC (Instantaneous Center of Curvature). The position control algorithm seeks to minimize the orientation error, $\theta_e = \alpha - \theta$, where α is the current angle to the target point and θ is the current orientation of the robot. At the same time, the robot tries to reduce the distance to the target point $(d \rightarrow 0)$. Figure 3 shows the block diagram of the control algorithm for this experiment, where the inner dashed square represents the controller and the outer dashed square represents the robot. Note that the position sensor is an IPS (Indoor Positioning System), which provides the absolute position and orientation of the robot Farias et al. (2019).

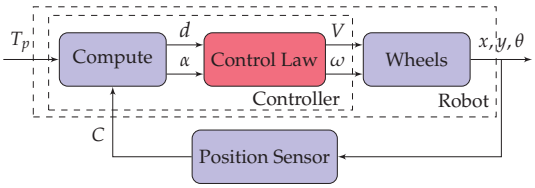

Fig. 3. Block diagram of the position control problem.

Equation (6) calculates the distance (d) and equation (7) calculates the angle to the target point (α). In both cases,

the values used for the calculation are the coordinates of $Tp(x_p, y_p)$ and $C(x_c, y_c)$. Note that both equations are implemented in the block *Compute*.

$$d = \sqrt{(y_p - y_r)^2 + (x_p - x_r)^2} \tag{6}$$

$$\alpha = \tan^{-1}\left(\frac{y_p - y_r}{x_p - x_r}\right) \tag{7}$$

Once the algorithm has the distance and angle to the destination point, it must obtain the corresponding angular and linear velocities to reach it, using the implementation of the block *Control Law*. This control law can be designed in different ways as was mentioned before. In this work, for simplicity we use the implementation described in Villela et al. (2004), which are the following equations:

$$V = \begin{cases} V_{max} & if\ |d| > K_r \\ d\left(\frac{V_{max}}{K_r}\right) & if\ |d| \leq K_r \end{cases} \tag{8}$$

$$\omega = \omega_{max} \sin(\alpha - \theta), \tag{9}$$

where V_{max} is the maximum linear velocity, K_r is the radius of a docking area (around the target point) and ω_{max} is the maximum angular velocity of the robot. Note that the linear velocity (V) is proportional to the distance inside the docking area, and it is otherwise saturated to its maximum value. The docking area allows the robot to reduce its linear velocity when it is near the destination point.

3.2 Mobile robot position control with RL

The main idea is to control the mobile robot using RL instead of a traditional control algorithm. To this end, the block *Control Law* has to be replaced for a trained RL model. During the training stage, this model learns how to obtain the velocities (V, ω) using distance (d) and angle (α) to the destination point. In this case, for simplicity in the learning stage, the model only learns about the angular velocity (ω) by using the error angle $(theta_e)$. The linear velocity (V) is kept constant at the maximum value while the robot is outside of the docking area.

As was mentioned before, this process is divided into two stages. During the first stage, the learning matrix Q is built. This matrix is composed of pairs (state, action), which are the states of the robot and its corresponding actions. The state, in this case, is an angular error (θ_e), and the action (control signal) is the angular velocity (ω) of the robot. The other state, distance d, is not used in the algorithm. Each pair (state, action) generates the rewards that compose the matrix Q. These rewards depend on the criterion that wants to be applied for the specific task to solve. In this approach, the criterion is to negatively penalize the deviations of the controlled signal, which are the big changes in the error angle of the robot, i.e. $\rho(x_k, u_k) = -abs(\theta_e)$. The actions have been divided into 20 regularly spaced values between $-\pi$ and π.

At the beginning of this stage, the algorithm finds in the matrix Q the current state of the robot using the current angular error. This state is used to determine if the robot advances randomly or if it advances taking into account the principle of exploration (uncharted territory) and exploitation (of current knowledge). The result is a value of the angular velocity selected from the actions, which is executed by the robot. Then, this process is repeated until a value defined by the user. The number of iterations depends on the obtained results and it is adjusted by trial and error. Note that the implementation of the algorithm considers the inclusion of disturbances by modification of the distance and angular error to the target point. The idea is to cover different scenarios to improve learning. These disturbances are supplied at the end of the learning stage. At this time the accumulated rewards are rewritten in the matrix Q using equation 4. Figure 4 shows an example of this stage after 5 million iterations.

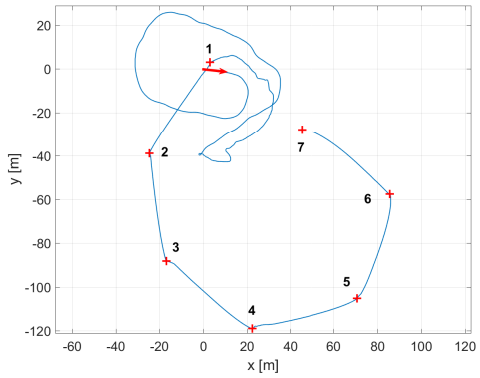

Fig. 4. Learning stage example.

The blue line represents the trajectory followed by the robot and the red crosses represent the destination points. The red arrow represents the initial position (0;0) and orientation of the robot. As can be seen, when the process starts, the robot has to reach point number 1 from its initial position at the origin. In this part of the learning process, its behavior is random because it is building the matrix Q using aleatory actions. After a big number of trials, the first target is reached. From point number 1 to point number 2, the built matrix Q is used to reach this destination point, which means that the robot uses the current knowledge to make the task. For the rest of the target points (3, 4, 5, 6 and 7), the process is repeated. Note that for these last targets the behavior of the robot is much more precise, which means that the robot has learned how to go from its current position to a predefined target. Figure 5 shows the resulting matrix Q after the learning stage of this example for 5 million iterations.

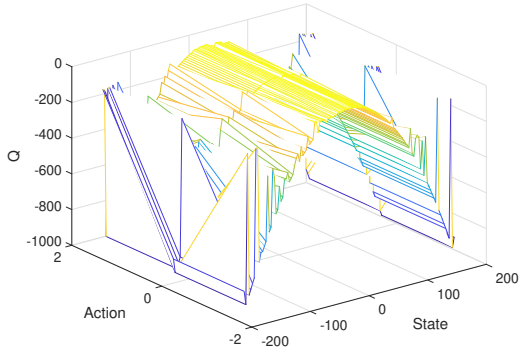

Fig. 5. Matrix Q at the end of the learning stage, States in degrees, and Action in radians per second.

4. RESULTS

This section shows some results of the implementation of the presented algorithm in simulation. The matrix Q is obtained in MATLAB after some iterations as was explained before. This matrix is exported into Python (Spyder-Anaconda). Spyder is connected to V-REP simulator to calculate the control law. The tests are carried out in V-REP using the KH4VREP library (Peralta et al. (2016); Farias et al. (2017)). Note that this library is a previous development of the authors. This library has been used in other developments obtaining reliable results, showing similar behavior in both, simulation and real platform (Farias et al. (2019)). In future versions of this work we will include results with the platform. Figure 6 shows the development environment.

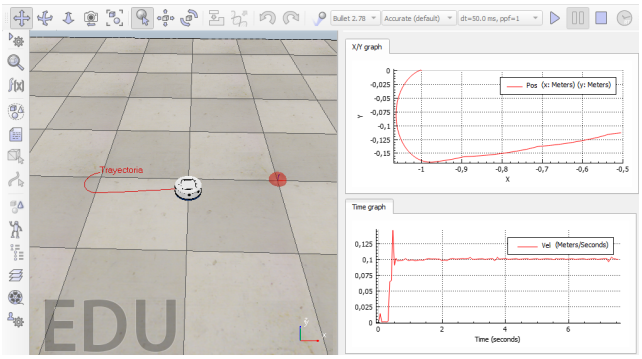

Fig. 6. Development environment.

4.1 Position control experience with RL

Figure 7 shows the results of the robot's position control experiment for different algorithms. In all the cases the initial conditions are the same. At the beginning of the experiment, the robot is at rest and its position is at point (-1;0). Its orientation is marked by the red arrow. The target point is marked by the red cross at the origin of the coordinates (0;0). The lines describe the trajectories followed by the robot for each algorithm.

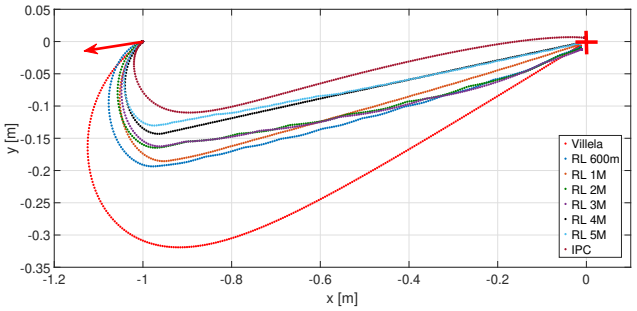

Fig. 7. Obtained trajectories for each control algorithm.

The red line represents the control algorithm described before in subsection 3.1 (Villela et al. (2004)). This control law shows the longer trajectory of all the algorithms studied in this work. On the other hand, the best performance is shown by the control law called IPC (Integral Position Control) (Fabregas et al. (2020)), which is represented by the brown line. The rest of the lines are the results of the implementation of the RL algorithm for different numbers of iterations in the calculation of the matrix Q. For example, RL600m represents the algorithm implemented with the matrix Q of 600 thousand iterations and the line RL 1M represents the algorithm with 1 million iterations. As can be seen, when the number of iterations is increased, the trajectory to the destination point is improved, which means that the robot describes a shorter trajectory to reach the destination point.

Note that these trajectories show a better performance than the Villela algorithm but it can't improve the IPC algorithm trajectory. Because in the IPC trajectory the linear velocity is also manipulated at the beginning of the experiment when the orientation error of the robot to the destination point is too big and the robot needs to turn more than to advance to get as soon as possible, the orientation to the goal. In the case of the RL algorithm, only the angular velocity is manipulated. That is why it never surpasses the IPC algorithm performance no matter the number of iteration in the calculation of the matrix Q. Figure 8 shows the distance to the destination point for the same experiments. The y-axis represents the distance in meters and the x-axis represents the time in seconds.

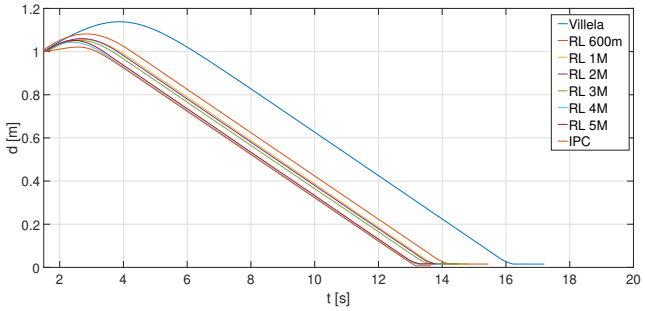

Fig. 8. Distance to the target for each control algorithm.

As can be seen, the trajectory that takes more time to reach the destination point (to make the distance equal to 0) is the Villela algorithm. While the IPC needs the shortest time to reach the destination. The rest of the trajectories show the expected behavior, decreasing the time to reach the destination when the number of iteration of the matrix Q is increased. To show the differences between the algorithms in a better way, different performance indexes for the distance to the destination have been compared, as it is shown in Table 1.

Table 1. Comparison between algorithms.

Index	Villela	RL600m	RL1M	RL3M	RL5M	IPC
IAE	10.18	7.91	7.47	7.24	6.90	6.75
ISE	8.96	6.39	5.89	5.63	5.30	5.10
ITSE	51.02	31.24	28.13	26.44	24.25	23.28
ITAE	68.21	46.45	42.84	40.95	38.03	36.96

As can be observed, the RL algorithm provides an improvement when the number of iterations in the calculation of the matrix Q is increased, converging to the optimal as time increases. This implies that the robot reaches the destination point with a shorter trajectory. Consequently, the different measurements of the integral error of the position over time are reduced.

5. CONCLUSION

In this paper, an algorithm to control the position of a wheeled mobile robot using Reinforcement Learning has been presented. This algorithm can be used at an academic institution to teach control engineering as an alternative paradigm to the traditional control approaches. The algorithm is divided into two parts: 1) Training stage and 2) Operational stage. In the first phase, the robot learns to reach a known destination point from its current position. During the learning process, the robot interacts with the environment and receives some rewards or punishment for its actions. This allows the robot to learn how to solve this task.

The results during the operational phase show that the obtained controller is capable of achieving the predefined task. The trajectories of the robot to the destination point are improved with the increase in the iterations on the learning stage. The trajectories never reach the IPC algorithm because in this case the linear velocity (V) is also manipulated. The main advantage of this algorithm is that the training stage is carried out automatically on a computer, in this way the damages to the robot during the learning time can be avoided. The main drawback is that this training stage can take a long time to finish. Note that you need a PC with high resources and performance because this stage can take millions of iterations to give good results.

In the near future, the algorithm will include the linear velocity of the robot (V) in the learning stage and the other remaining state, the distance (d). This can make the learning process more complex and will increase the training time, but in turn, it can increase the efficiency and accuracy of the resulting model. Once both velocities and both states are included in the model, the next step will be to implement the control law in the robot Khepera IV that we have in our laboratory, to compare the results with previous control law (IPC) that has been designed and tested.

ACKNOWLEDGEMENTS

This work was supported in part by the Spanish Ministry of Economy and Competitiveness under Projects CICYT RTI2018-094665-B-I00 and ENE2015-64914-C3-2-R and by the Chilean Ministry of Education under Project FONDECYT 1191188.

REFERENCES

Busoniu, L., Babuska, R., De Schutter, B., and Ernst, D. (2017). *Reinforcement learning and dynamic programming using function approximators*. CRC press.

Caceres, C., Rosario, J.M., and Amaya, D. (2017). Approach of kinematic control for a nonholonomic wheeled robot using artificial neural networks and genetic algorithms. In *2017 International Conference and Workshop on Bioinspired Intelligence (IWOBI)*, 1–6.

Dumitrascu, B., Filipescu, A., and Mînzu, V. (2011). Backstepping control of wheeled mobile robots. In *15^{th} International Conference on System Theory, Control and Computing, ICSTCC 2011*.

Fabregas, E., Farias, G., Aranda-Escolástico, E., Garcia, G., Chaos, D., Dormido-Canto, S., and Dormido, S. (2020). Simulation and experimental results of a new control strategy for point stabilization of nonholonomic mobile robots. *IEEE Transactions on Industrial Electronics*, 67(8), 6679 – 6687.

Fabregas, E., Farias, G., Dormido-Canto, S., Guinaldo, M., Sánchez, J., and Dormido Bencomo, S. (2016). Platform for teaching mobile robotics. *Journal of Intelligent & Robotic Systems*, 81(1), 131–143.

Fabregas, E., Farias, G., Peralta, E., Sanchez, J., and Dormido, S. (2017). Two mobile robots platforms for experimentation: Comparison and synthesis. In *Proceedings of the 14^{th} International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, volume 2, 439–446. SciTePress.

Farias, G., Fabregas, E., Peralta, E., Vargas, H., Dormido-Canto, S., and Dormido, S. (2019). Development of an easy-to-use multi-agent platform for teaching mobile robotics. *IEEE Access*, 7, 55885–55897.

Farias, G., Fabregas, E., Peralta, E., Torres, E., and Dormido, S. (2017). A Khepera IV library for robotic control education using V-REP. In *IFAC-PapersOnLine 20^{th} IFAC World Congress*, volume 50, 9150–9155.

González, R., Fiacchini, M., Alamo, T., Guzmán, J.L., and Rodriguez, F. (2010). Adaptive control for a mobile robot under slip conditions using an lmi-based approach. *European Journal of Control*, 16(2), 144 – 155.

Jaksch, T., Ortner, R., and Auer, P. (2010). Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr), 1563–1600.

Mohareri, O., Dhaouadi, R., and Rad, A.B. (2012). Indirect adaptive tracking control of a nonholonomic mobile robot via neural networks. *Neurocomputing*, 88, 54 – 66. Intelligent and Autonomous Systems.

Omrane, H. and Masmoudi, M. (2016). Fuzzy logic based control for autonomous mobile robot navigation. *Computational intelligence and neuroscience*, 2016.

Peralta, E., Fabregas, E., Farias, G., Vargas, H., and Dormido, S. (2016). Development of a Khepera IV library for the V-REP simulator. *IFAC-PapersOnLine*, 49(6), 81–86. 11^{th} IFAC Symposium on Advances in Control Education ACE 2016.

Rezaee, A. (2017). Model predictive controller for mobile robot. *Transactions on Environment and Electrical Engineering*, 2, 17.

Siegwart, R., Nourbakhsh, I., and Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*. MIT press.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tzafestas, S.G. (2018). Mobile robot control and navigation: A global overview. *Journal of Intelligent & Robotic Systems*, 91(1), 35–58.

Villela, V., Parkin, R., López, M., and Dorador, J. (2004). A wheeled mobile robot with obstacle avoidance capability. *Tecnología Y Desarrollo*, 1(5), 159–166.

Watkins, C. (1989). *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge, King's Parade, Cambridge CB2 1ST, United Kingdom.

Watkins, C. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4), 279–292.

White, C.C. (2001). *Markov decision processes*. Springer US, Boston, MA.